Control software analysis, Part II
Closed-loop properties

Eric Feron [*] Fernando Alegre [†]

December 10, 2008

Abstract: The analysis and proper documentation of the properties of closed-loop control software presents many distinct aspects from the analysis of the same software running open-loop. Issues of physical system representations arise, and it is desired that such representations remain independent from the representations of the control program. For that purpose, a concurrent program representation of the plant and the control processes is proposed, although the closed-loop system is sufficiently serialized to enable a sequential analysis. While dealing with closed-loop system properties, it is also shown by means of examples how special treatment of nonlinearities extends from the analysis of control specifications to code analysis.

1 Closed-loop software system analysis

The first part of this report essentially dealt with the analysis of open-loop control systems, that is, the analysis of control algorithms without regards for their stabilizing effect on the plant: In [FA08], we were concerned with verifying and quantifying the boundedness of all computed quantities in the controller, regardless of the actual performance of the closed-loop system. In that regard, the analysis is similar to that described in [CCF$^+$05], although our work is much more focused on control algorithms and, unlike the work in [CCF$^+$05], it ignores the many other software components present in a fully developed avionics system. In the present document, we focus on the analysis and verification of the system's closed-loop performance, beginning with closed-loop system stability. One of the elements of this report is the modeling, by means of an example, of closed-loop control systems as two concurrent processes, where one process represents the real-time computer program and the other represents the system being controlled. The concurrent representation of these processes does not make their analysis more complicated, and it provides the engineer with greater flexibility by making the choice of controller representations and implementations independent from the representation of the plant being controlled.

[*]Dutton/Ducoffe Professor of Aerospace Software Engineering, Georgia Institute of Technology. `feron@gatech.edu`

[†]College of Computing, Georgia Tech. `fernando@cc.gatech.edu`

Again, the best way to perform the closed-loop system analysis is to begin with specification-level closed-loop system analysis, and then to move on to incorporate more detailed controller implementations. This is the process illustrated in this report.

1.1 Specification-level analyses: Stability

The analysis of the specifications of the lead-lag controller designed and presented in [FA08] begins with Figure 1. In that figure, the controlled system is represented by a continuous differential equation, while the controller is a discrete-time system that interacts with the continuous system by means of a signal sampler at the input and a zero-order hold at the output. The dynamics of the controller may be written

$$\begin{aligned} x_{c,k+1} &= A_c x_{c,k} + B_c\,\mathbf{SAT}(y_k) \\ u_k &= C_c x_{c,k} + D_c\,\mathbf{SAT}(y_k) \end{aligned}$$

with

$$A_c = \begin{bmatrix} 0.4999 & -0.0500 \\ 0.0100 & 1.0000 \end{bmatrix},\ B_c = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$C_c = [564.48\ \ 0]\,,\ \text{and}\ D_c = -1280.$$

It is well-known from control theory [FP80] that the analysis of such a closed-loop, sampled-data system can be performed by replacing the plant, the zero-order hold and the sampler by the discrete-time system

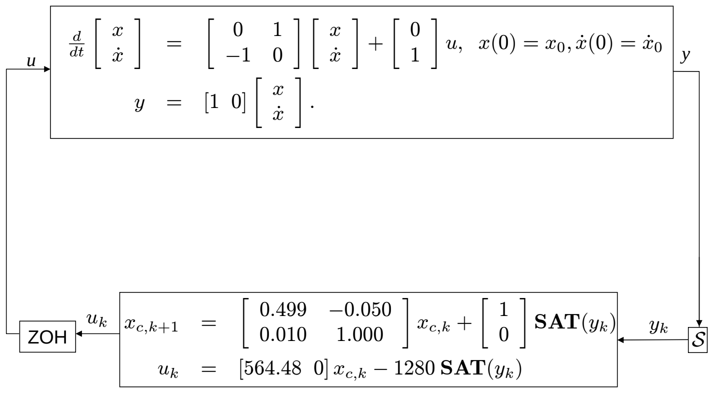
$$\begin{aligned} x_{p,k+1} &= A_p x_{p,k} + B_p u_k \\ y_k &= C_p x_{p,k} \end{aligned}$$

with

$$A_p = \begin{bmatrix} 1.0000 & 0.0100 \\ -0.0100 & 1.0000 \end{bmatrix},\ B_p = \begin{bmatrix} 0.00005 \\ 0.01 \end{bmatrix}$$

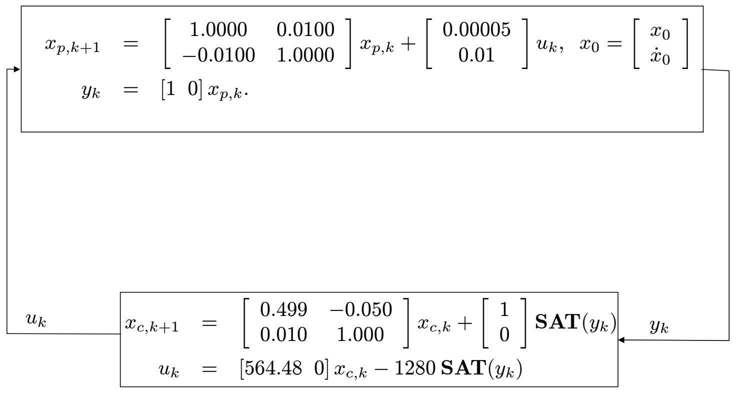
and $C_p = [1\ \ 0]$. Moreover, $x_{p,k} = x_p(0.01k)$, where $k \in \mathbf{N}$. Then the analysis of the system shown in Fig. 1 is equivalent to the analysis of the discrete-time system shown in Fig. 2. The stability analysis of this discrete-time system is made more difficult than that of the controller alone [FA08] because of the presence of a saturation nonlinearity aimed at limiting the range of sensor inputs to the control system. This nonlinearity may be accounted for in the overall system stability analysis by using a variety of computational techniques, such as those described in [BEFB94]. In particular one of the available techniques consists of computing a quadratic Lyapunov function for the system. It is, for example, possible to show that the quadratic function

$$V(x) = \begin{bmatrix} x_c \\ x_p \end{bmatrix}^T P \begin{bmatrix} x_c \\ x_p \end{bmatrix},\ \text{with}\ P = \begin{bmatrix} 0.2205 & 0.0188 & -0.0750 & 0.0177 \\ 0.0188 & 0.4736 & 0.0535 & 0.0015 \\ -0.0750 & 0.0535 & 0.1012 & -0.0049 \\ 0.0177 & 0.0015 & -0.0049 & 0.0015 \end{bmatrix}$$



$\frac{d}{dt}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u, \quad x(0) = x_0, \dot{x}(0) = \dot{x}_0$
$y = [1 \;\; 0]\begin{bmatrix} x \\ \dot{x} \end{bmatrix}.$
u
y
$\mathcal{S}$
y_k
$x_{c,k+1} = \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k)$
$u_k = [564.48 \;\; 0]\, x_{c,k} - 1280\, \mathbf{SAT}(y_k)$
u_k
ZOH

Figure 1: Feedback system



$x_{p,k+1} = \begin{bmatrix} 1.0000 & 0.0100 \\ -0.0100 & 1.0000 \end{bmatrix} x_{p,k} + \begin{bmatrix} 0.00005 \\ 0.01 \end{bmatrix} u_k, \quad x_0 = \begin{bmatrix} x_0 \\ \dot{x}_0 \end{bmatrix}$
$y_k = [1 \;\; 0]\, x_{p,k}.$
u_k
$x_{c,k+1} = \begin{bmatrix} 0.499 & -0.050 \\ 0.010 & 1.000 \end{bmatrix} x_{c,k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{SAT}(y_k)$
$u_k = [564.48 \;\; 0]\, x_{c,k} - 1280\, \mathbf{SAT}(y_k)$
y_k

Figure 2: Equivalent discrete system

decays along all trajectories $(x_{c,k}, x_{p,k})_{k=1,2,...}$ for initial conditions where the controller is at rest ($x_{c,0} = 0$) and the initial state $x_{p,0}$ of the mechanical system lies inside the ellipse $\mathcal{E}_Q$ with

$$Q = \begin{bmatrix} 0.1012 & -0.0049 \\ -0.0049 & 0.0015 \end{bmatrix} \text{ and } \mathcal{E}_Q = \{y \in \mathbf{R}^2 \mid y^T Q y \leq 1\} \tag{1}$$

The mechanisms used to reach that conclusion rely on standard stability considerations, most notably absolute stability theory: Consider the closed-loop system dynamics

$$x_{k+1} = Ax_k + B\mathbf{SAT}(Cx_k),$$

with $x = [x_c^T \ \ x_p^T]^T$,

$$A = \begin{bmatrix} A_c & 0 \\ B_pC_c & A_p \end{bmatrix} = \begin{bmatrix} 0.4990 & -0.0500 & 0 & 0 \\ 0.0100 & 1.0000 & 0 & 0 \\ 0.0282 & 0 & 1.0000 & 0.0100 \\ 5.6448 & 0 & -0.0100 & 1.0000 \end{bmatrix},$$

$$B = \begin{bmatrix} B_c \\ B_pD_c \end{bmatrix}, \text{ and } C = [0 \ \ C_p].$$

Consider the set of x's such that $x^T P x < 1$. It is easy to show that (i) the set of all $x = [0 \ \ x_p^T]^T$ such that $x_p \in \mathcal{E}_Q$ is contained in $\mathcal{E}_P$, and (ii) that for all $x \in \mathcal{E}_P$, $(\mathbf{SAT}(Cx) - 0.2Cx)(\mathbf{SAT}(Cx) - Cx) \leq 0$. Introducing $y_{c,k} = \mathbf{SAT}(Cx_k)$, it is therefore sufficient to show that

$$x_{k+1}^T P x_{k+1} - x_k^T P x_k \leq 0$$

whenever $(y_{c,k} - 0.2Cx_k)(y_{c,k} - Cx_k) \leq 0$. An equivalent condition is

$$\begin{bmatrix} x \\ y_c \end{bmatrix}^T \left(\begin{bmatrix} A^T \\ B^T \end{bmatrix} P[A\ B] - \begin{bmatrix} I \\ 0 \end{bmatrix} P\,[I\ 0] \right) \begin{bmatrix} x \\ y_c \end{bmatrix} \leq 0$$

whenever $(y_c - 0.2Cx)(y_c - Cx) \leq 0$. Using the well-known $\mathcal{S}$-procedure [AG64], a sufficient condition is the existence of $\lambda > 0$ such that

$$\begin{bmatrix} A^T \\ B^T \end{bmatrix} P[A\ B] - \begin{bmatrix} I \\ 0 \end{bmatrix} P\,[I\ 0] - \lambda \begin{bmatrix} 0.2C^TC & -0.6C^T \\ -0.6C & 1 \end{bmatrix} \leq 0. \tag{2}$$

The latter inequality can be easily shown to be true for $\lambda = 6.76$. It must be kept in mind that all statements related to numerical objects and computations are made under the assumption that computations on reals are exact. This assumption is obviously false and must be eventually addressed when implementing the operational software verification tools that may arise from this paper, using for example techniques developped by Goubault [Gou01]. We are now interested in showing how this proof of (local) closed-loop stability may be translated at the code and system level. To be more precise, we will show how invariance of the ellipsoid $\mathcal{E}_P$ can be exploited to develop a proof of proper

```
% Controller Dynamics
1c:  Ac = [0.4990, -0.0500; 0.0100,1.0000];
2c:  Cc=[564.48, 0];
3c:  Bc=[1;0]; D = -1280;
4c:  xc = zeros(2,1);
5c:  receive(y);
6c:  while (1)
7c:    yc = max(min(y,1),-1);
8c:    u = Cc*xc + Dc*yc;
9c:    xc = Ac*xc + Bc*yc;
10c:   send(u);
11c:  receive(y);
12c:  end
```

```
% Plant Dynamics
1p:  Ap = [1.0000,0.0100;-0.0100,1.0000];
2p:  Cp=[1,0];
3p:  Bp = [0.00005; 0.01];
4p:  while (1)
5p:     y = Cp*xp;
6p:     send(y);
7p:     receive(u);
8p:     xp = Ap*xp + Bp*u;
9p:  end;
```

Table 1: Concurrent representation of plant and controller implementations

behavior, such as stability and performance, for the computer program that implements the controller, as it interacts with the physical system. Unlike the developments relative to controller open-loop properties, this proof necessarily involves the presence of the controlled artifact. It would not make sense for the system to be represented at various degrees of computer program implementations since these do not have any physical meaning. Thus we choose to represent the plant and the computer program as two concurrent programs, as shown in Table 1. The computer program representation of the mechanical system will remain invariant, written in a high-level language like MATLAB, while the representation of the controller will be allowed to evolve to reflect the various stages of its implementation, without changing the representation of the physical system. We assume the two programs communicate by means of `send / receive` commands. It is assumed that the `receive` command is blocking, that is, the program execution cannot proceed until the variable of interest has been received. While the programs in Table 1 are purely event-driven, it is possible to modify the algorithmic description of the plant to account for the presence of time. The question of establishing proofs of stability of the closed-loop system at the code level is necessarily tied to understanding the behavior of the controller and that of the plant. The entire state-space then consists of the direct sum of the controller's state-space and that of the plant. We now propose to leverage the approach developed in the first part of this report [FA08] to document the corresponding system of two concurrent programs. This documentation can be extended to other representations of the control program implementation: Table 2 shows one such implementation of the controller in pseudo-C. One of the interesting aspects of the processes in this report is their concurrency, which can make the structure of the state transitions rather complicated. However, a close inspection of the programs reveals a relatively simple transition structure. Our investigation thus begins with a forward analysis of both programs. Since

```
/* variable declarations */
double Ac[2][2];
double Cc[2];
double Bc[2];
double Dc;
double xc[2], yc,y,u;
int flags, flaga;

/* aux vars */
int i,j;
double xc_new[2];

/* code */
Ac[0][0]=0.4990; Ac[0][1]=-0.0500; Ac[1][0]=0.0100;Ac[1][1]=1.0000;
Cc[0]=564.48;Cc[1]=0.0;
Bc[0]=1.;Bc[1]=0.;
Dc=-1280.;
xc[0]=0.;xc[1]=0.;
receive(y);
while(1) {
  yc = (y > 1. ? 1. : y) < -1. ? -1. : y;
  u = 0.0;
  for(i=0;i<2;i++) u += Cc[i]*xc[i];
  u += Dc*yc;
  for(i=0;i<2;i++) {
    xc_new[i] = 0.0;
    for(j=0;j<2;j++) xc_new[i] += Ac[i][j]*xc[j];
    xc_new[i] += Bc[i]*yc;
  }
  for(i=0;i<2;i++) xc[i] = xc_new[i];
  send(u);
  receive{y}
}
```

Table 2: Implementation of controller in C

only local stability has been proven from the program specifications, we may not expect any better result by inspecting and documenting the code. Declared but non-initialized variables v will be assumed to belong to the empty set, that is $v \in \perp$. Likewise, the set of all possible values will be denoted $\top$, and a variable v that may take any possible value will be characterized as $v \in \top$. Since both programs run concurrently, tracking state values and the sets they belong to may be somewhat challenging. The only variable whose value is assumed to be initialized prior to the plant execution is the initial plant state x_p. For the purpose of stability analysis, we assume that the state x_p is initially within the ellipsoid $\mathcal{E}_Q$, where Q is defined in (1). Thus, prior to the initiation of the plant process, and using insight from the system specification analysis, we form the condition $\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$. The first three lines of the plant dynamics `1p, 2p, 3p` do not change this condition, and so they may be commented as

$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$
```
1p:  Ap = [1.0000,0.0100;-0.0100,1.0000];
```
$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$
```
2p:  Cp=[1,0];
```
$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$
```
3p:  Bp = [0.00005; 0.01];
```
$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$

Line `4p:  while (1)`, together with line `9p:  end;` defines the main loop of the plant dynamics. We use the requirements analysis to postulate the following set of pre- and post-conditions

$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$
```
4p:  while (1)
```
$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$

$\vdots$

$\{(x_c, x_p) \in \mathcal{E}_P,\ (y, y_c, u) \in \top\}$
```
9p:  end
```
$\{false\}$

The last assertion $\{false\}$ simply indicates a never ending loop; this should be expected from a dynamical system that never ceases to execute. In the following, and in order to simplify the notation, we will omit to mention the variables whose value could be arbitrary (that is, the variables whose values belong to $\top$). With this convention, the previous set of commented lines becomes

$\{(x_c, x_p) \in \mathcal{E}_P\}$
`4p:  while (1)`
$\{(x_c, x_p) \in \mathcal{E}_P\}$
$\vdots$
$\{(x_c, x_p) \in \mathcal{E}_P\}$
`9p:  end`
$\{false\}$.

The following line, `5p:  y = Cp*xp` assigns a value to the variable y. The corresponding pair of conditions is therefore

$\{(x_c, x_p) \in \mathcal{E}_P\}$
`5p:  y = Cp*xp`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$,

with

$$\mathcal{G}_R = \left\{ x \in \mathbf{R}^n \;\middle|\; \begin{bmatrix} 1 & x^T \\ x & R \end{bmatrix} \geq 0 \right\}.$$

The next line, `6p:  send(y);` does not affect the state variables of interest and therefore becomes

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`6p:  send(y)`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$

At this point, the execution of the plant process is blocked by the `receive` command in line `7p`. We now turn our attention to the controller code. The first four lines of the controller code `1c, 2c, 3c, 4c`, are commented using the statement $\{(x_c, x_p, y) \in \mathcal{G}_R,\ (y_c, u) \in \top\}$ or, in short, $\{(x_c, x_p, y) \in \mathcal{G}_R\}$. Thus we get

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`1c:  Ac = [0.4990, -0.0500; 0.0100,1.0000];`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`2c:  Cc=[564.48 0];`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`3c:  Bc=[1; 0]; D = -1280;`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$,
`4c:  xc = zeros(2,1);`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$.

At this point, the execution of the controller program is halted by the command `5c:  receive(y)` and can proceed only after the plant has executed line `6p`. Valid pre- and post-conditions are

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`5c:  receive(y);`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$

We then encounter line `6c: while (1)` which, together with line `12c: end` defines the main loop of the controller. We postulate the following pre- and post-conditions for these instructions

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`6c: while (1)`
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
$\vdots$
$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`12c: end`
$\{false\}$.

The next line `7c: yc = max(min(y,1),-1)`, applies the nonlinear saturation operator to y. Common systems practice (as well as the methods used to compute P in the first place) suggests that this nonlinear relation may be accurately captured by the quadratic inequality

$$(y_c - y)(y_c - 0.2y) \leq 0,$$

also named a *sector bound* on the saturation operator. This sector bound is not true in general, but it holds for all variables in $\mathcal{G}_R$. This bound may also be expressed in matrix form

$$\begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix}^T \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & -0.6 \\ 0 & 0 & -0.6 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} = \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix}^T T \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} \leq 0. \qquad (3)$$

Thus one possible set of pre- and post- conditions for line 7c is

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
`7c: yc = max(min(y,1),-1)`
$$\left\{ (x_c, x_p, y) \in \mathcal{G}_R, \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix}^T T \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} \leq 0 \right\}$$

However, following the principles outlined in [FA08], we seek to capture all variables within a single quadratic constraint, for the purpose of making proof checking simple and mirroring the usage of the $\mathcal{S}$-procedure to establish stability proofs at the specification level. To do so, we rely on the following lemma:

Lemma: Assume the real vector variables z and w satisfy the constraints

$$\begin{bmatrix} 1 & z^T \\ z & U \end{bmatrix} \geq 0$$

for a given $U = U^T$ and

$$\begin{bmatrix} z \\ w \end{bmatrix}^T T \begin{bmatrix} z \\ w \end{bmatrix} \leq 0$$

for $T = T^T$. Then for any real μ, we have

$$\begin{bmatrix} z \\ w \end{bmatrix} \in \mathcal{G}_V, \;\; V = \left(\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} - \mu \begin{bmatrix} U & 0 \\ 0 & I \end{bmatrix} T \right)^{-1} \begin{bmatrix} U & 0 \\ 0 & I \end{bmatrix}$$

whenever V, defined as such, exists and is positive definite.

Proof: The proof is simple and therefore omitted.

Applying this lemma with $U = R$, T defined as in (3) and $\mu = -\lambda = -6.76$ yields the inequality

$$\begin{bmatrix} 1 & [x_c^T \; x_p^T \; y \; y_c] \\ \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} & V \end{bmatrix} \geq 0.$$

and the pre- and post-conditions

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$
```
7c:  yc = max(min(y,1),-1);
```
$\{(x_c, x_p, y, y_c) \in \mathcal{G}_V\}$

At this point, the variable y is not of use anymore and we may therefore release it. Thus we replace the post-condition $\{(x_c, x_p, y, y_c) \in \mathcal{G}_V\}$ of line 7c by the post condition $\{(x_c, x_p, y_c) \in \mathcal{G}_W\}$, with

$$W = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix} V \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & I \end{bmatrix}^T$$

The assignment `8c:  u = Cc*xc + Dc*yc` is then easily commented as

$\{(x_c, x_p, y_c) \in \mathcal{G}_W\}$
```
8c:  u = Cc*xc + Dc*yc
```
$\{(x_c, x_p, u, y_c) \in \mathcal{G}_X\}$

with

$$X = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C_c & 0 & D_c \\ 0 & 0 & 1 \end{bmatrix} W \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C_c & 0 & D_c \\ 0 & 0 & 1 \end{bmatrix}^T.$$

The next line of the controller loop `9c:  xc = Ac*xc + Bc*yc` can be treated similarly to line 8c, to yield the triple

$\{(x_c, x_p, u, y_c) \in \mathcal{G}_X\}$
```
9c:  xc = Ac*xc + Bc*yc
```
$\{(x_c, x_p, u) \in \mathcal{G}_Y\}$

with

$$Y = \begin{bmatrix} Ac & 0 & 0 & Bc \\ 0 & I & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} X \begin{bmatrix} Ac & 0 & 0 & Bc \\ 0 & I & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}^T .$$

Note that the variable y_c is not used anymore and has therefore been released. Line `10c` does not influence the variables and therefore can be commented as

$\{(x_c, x_p, u) \in \mathcal{G}_Y\}$
`10c:  send(u)`
$\{(x_c, x_p, u) \in \mathcal{G}_Y\}$

The only remaining line in the controller loop is then line `11c:  receive(y)`, which blocks the controller loop until a new value of y is available.

At that point, it becomes important to look again at the plant process, which restarts at line `7p:  receive(u)`. A necessary post-condition for this line is $\{(x_c, x_p, u) \in \mathcal{G}_Y\}$, but no pre-condition can be clearly extracted. Line 7p will therefore be only partially instrumented, replacing the pre-condition by the symbol $\vdots$. In addition, we indicate between brackets which line execution in the controller code (line 10c) unlocks line 7p, yielding the "triple"

$\vdots$
`7p:  receive(u) [10c]`
$\{(x_c, x_p, u) \in \mathcal{G}_Y\}$.

Consider then line `8p:   xp = Ap*xp + Bp*u;`. This assignment is properly documented with pre- and post- conditions to yield

$\{(x_c, x_p, u) \in \mathcal{G}_Y\}$
`8p:   xp = Ap*xp + Bp*u;`
$\{(x_c, x_p) \in \mathcal{G}_Z\}$

where

$$Z = \begin{bmatrix} I & 0 & 0 \\ 0 & A_p & B_p \end{bmatrix} Y \begin{bmatrix} I & 0 & 0 \\ 0 & A_p & B_p \end{bmatrix}^T ,$$

and u has been released.

At that point, the plant dynamics meets line `9p:  end`. That line has already been instrumented with candidate pre-and post conditions, which therefore must be shown to be compatible with the post-condition of line 9p. More precisely, we must show

$$(x_c, x_p) \in \mathcal{G}_Z \Rightarrow (x_c, x_p) \in \mathcal{E}_P. \tag{4}$$

Numerical computations show that this is indeed true (modulo floating point operation arithmetic errors). While the entire loop describing the plant dynamics has been investigated, such is not the case of the controller dynamics, whose lines 11c and 12c have not been executed yet. We therefore need to keep tracking the execution of the two programs. While the control program is still blocked at line 11c, the plant dynamics loop sends the program execution to line 4p, 5p, and 6p, where the pre- and post- conditions are already established and coherent. Line 7p then blocks any further propagation of the plant dynamics, while the controller program is now allowed to proceed past line 11c, for which a valid post-condition is therefore $\{(x_c, x_p, y) \in \mathcal{G}_R\}$ and we will instrument as follows

$\vdots$

```
11c:  receive(y) [6p]
```

$\{(x_c, x_p, y) \in \mathcal{G}_R\}$

which is compatible with the following line, 12c, which has been already documented.

The resulting commented programs then look like those shown in Table 3.

2 Sector-bounded nonlinearities and stability analysis

Following similar developments in [FA08], there are apparent differences between the stability analysis performed in the requirements-level stability analysis, summarized by Eq. (2), and the forward analysis performed afterwards. Namely, we want to show that the invariance condition (4) is satisfied if and only if the condition (2) is satisfied. For that purpose, we begin with the forward analysis of the system, beginning at line `7c:  yc = max(min(y,1),-1)`. The assertion

$$\left\{ (x_c, x_p, y) \in \mathcal{G}_R, \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix}^T T \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} \leq 0 \right\}$$

with

$$R = \begin{bmatrix} P^{-1} & P^{-1}C^T \\ CP^{-1} & CP^{-1}C^T \end{bmatrix}, \text{ and } T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & -0.6 \\ 0 & 0 & -0.6 & 1 \end{bmatrix}$$

implies the assertion

$$\begin{bmatrix} 1 & [x_c^T \ x_p^T \ y \ y_c] \\ \begin{bmatrix} x_c \\ x_p \\ y \\ y_c \end{bmatrix} & V \end{bmatrix} \geq 0,$$

with

$$V = \left(\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} - \mu \begin{bmatrix} R & 0 \\ 0 & I \end{bmatrix} T \right)^{-1} \begin{bmatrix} R & 0 \\ 0 & I \end{bmatrix}.$$

```
% Controller Dynamics
{$(x_c, x_p, y) \in \mathcal{G}_R$}
1c:  Ac = [0.4990, -0.0500; 0.0100,1.0000];
{$(x_c, x_p, y) \in \mathcal{G}_R$}
2c:  Cc=[564.48, 0];
{$(x_c, x_p, y) \in \mathcal{G}_R$}
3c:  Bc=[1;0]; D = -1280;
{$(x_c, x_p, y) \in \mathcal{G}_R$}
4c:  xc = zeros(2,1);
{$(x_c, x_p, y) \in \mathcal{G}_R$}
5c:  receive(y)
{$(x_c, x_p, y) \in \mathcal{G}_R$}
6c:  while (1)
{$(x_c, x_p, y) \in \mathcal{G}_R$}
7c:  yc = max(min(y,1),-1)
{$(x_c, x_p, y, y_c) \in \mathcal{G}_V$}
skip
{$(x_c, x_p, y_c) \in \mathcal{G}_W$}
8c:  u = Cc*xc + Dc*yc
{$(x_c, x_p, u, y_c) \in \mathcal{G}_X$}
9c:  xc = Ac*xc + Bc*yc
{$(x_c, x_p, u) \in \mathcal{G}_Y$}
10c:  send(u)
{$(x_c, x_p, u) \in \mathcal{G}_Y$}
⋮
11c:  receive(y) [6p]
{$(x_c, x_p, y) \in \mathcal{G}_R$}
12c:  end
{$false$}.
```

```
% Plant Dynamics
{$(x_c, x_p) \in \mathcal{E}_P$}
1p:  Ap = [1.0000,0.0100;-0.0100,1.0000];
{$(x_c, x_p) \in \mathcal{E}_P$}
2p:  Cp=[1,0];
{$(x_c, x_p) \in \mathcal{E}_P$}
3p:  Bp = [0.00005; 0.01];
{$(x_c, x_p) \in \mathcal{E}_P$}
4p:  while (1)
{$(x_c, x_p) \in \mathcal{E}_P$}
5p:  y = Cp*xp
{$(x_c, x_p, y) \in \mathcal{G}_R$}
6p:  send(y);
{$(x_c, x_p, y) \in \mathcal{G}_R$}
⋮
7p:  receive(u); [10c]
{$(x_c, x_p, u) \in \mathcal{G}_Y$}
8p:   xp = Ap*xp+ Bp*u;
{$(x_c, x_p) \in \mathcal{G}_Z$}
skip
{$(x_c, x_p) \in \mathcal{E}_P$}
9p:  end
{$false$}
```

Table 3: Commented closed-loop control program

Long and somewhat tedious computations indicate that W, where W is obtained by erasing the row and column of V corresponding to the the variable y, satisfies

$$\begin{aligned} W &= \begin{bmatrix} P + 0.2\mu C^T C & -0.6\mu C^T \\ -0.6\mu C & -\mu \end{bmatrix}^{-1} \\ &= \begin{bmatrix} \tilde{P}^{-1} & 0.6\tilde{P}^{-1}C^T \\ 0.6C\tilde{P}^{-1} & -\dfrac{1}{\mu} + 0.36\mu C\tilde{P}^{-1}C^T \end{bmatrix} \\ &= \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix} \begin{bmatrix} \tilde{P}^{-1} & 0 \\ 0 & -\dfrac{1}{\mu} \end{bmatrix} \begin{bmatrix} I & 0.6C^T \\ 0 & 1 \end{bmatrix}, \end{aligned}$$

with $\tilde{P} = P - 0.16\mu C^T C$. Following the forward analysis and substituting matrix expressions for their values eventually yield

$$Z = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix} \begin{bmatrix} \tilde{P}^{-1} & 0 \\ 0 & -\dfrac{1}{\mu} \end{bmatrix} \begin{bmatrix} I & 0.6C^T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix}^T .$$

The invariance condition (4) therefore becomes

$$\begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix} \begin{bmatrix} \tilde{P}^{-1} & 0 \\ 0 & -\dfrac{1}{\mu} \end{bmatrix} \begin{bmatrix} I & 0.6C^T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix}^T \leq P^{-1}$$

Using Schur complements a first time yields the inequality

$$\left[\begin{array}{c|c} -P^{-1} & \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix} \\ \hline [30pt] \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix}^T \begin{bmatrix} A & B \end{bmatrix}^T & \begin{matrix} -\tilde{P} & 0 \\ 0 & \mu \end{matrix} \end{array}\right] \leq 0.$$

Using Schur complements a second time then yields the inequality

$$\begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix}^T \begin{bmatrix} A & B \end{bmatrix}^T P \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I & 0 \\ 0.6C & 1 \end{bmatrix} - \begin{bmatrix} \tilde{P} & 0 \\ 0 & -\mu \end{bmatrix} \leq 0.$$

Multiplying this inequality to the left and right by

$$\begin{bmatrix} I & 0 \\ -0.6C & 1 \end{bmatrix}^T \text{ and } \begin{bmatrix} I & 0 \\ -0.6C & 1 \end{bmatrix},$$

respectively, yields

$$\begin{bmatrix} A & B \end{bmatrix}^T P \begin{bmatrix} A & B \end{bmatrix} - \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} + \mu \begin{bmatrix} 0.2C^T C & -0.6C^T \\ -0.6C & 1 \end{bmatrix} \leq 0,$$

which is the same as (2), with $\mu = -\lambda$.

3 Conclusions

This report describes an approach to documenting control programs, whereby the control program code is annotated with logical expressions describing the set of reachable program states. This approach constitutes the application of the Floyd-Hoare paradigm to control programs. It is shown that the the extensive domain knowledge gathered by control theory about control system specifications is readily applicable to develop stability and performance proofs of the corresponding control programs. Some system elements, such as sector-bounded nonlinearities, can be handled via forward analysis, and this analysis was shown to be equivalent to well known results in absolute stability theory.

The analyses discussed in this report may be used in several different contexts: First, they may be used in an autocoding environment, whereby diagram-based specification languages such as Simulink can be autocoded in a target language such as C to automatically produce software with extensive, inlined proofs of software stability and performance. Such a documented software may then be easily verified independently by a proof checker.

Second, the analyses described in this paper may also be used to produce stability and poerformance proofs of undocumented software, even if little or no information is available about the software specifications. Producing such proofs becomes an integral part of the software verification process. Although time-consuming, this process is the only available option for much of the existing body of real-time control software. A particular case of interest is concerned with autocoded sofftware, where prior understanding of the autocoding process may help speed up the verification task.

Acknowledgements

This report was prepared under partial support from the National Science Foundation under grant CSR/EHS 0615025, NASA under cooperative agreement NNX08AE37A, and the Dutton-Ducoffe professorship at Georgia Tech.

References

[AG64] M. A. Aizerman and F. R. Gantmacher. *Absolute stability of regulator systems.* Information Systems. Holden-Day, San Francisco, 1964.

[BEFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *SIAM Studies in Applied Mathematics.* SIAM, 1994.

[CCF$^+$05] P. Cousot, R. Cousot, J. Feret, A. Miné, D. Monniaux, L. Mauborgne, and X. Rival. The ASTRÉE analyzer. In *In S. Sagiv (Ed.), Programming Languages and Systems, 14th European Symposium on Programming, ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005,* *Lecture Notes in Computer Science 3444*, pages 21–30, Edinburgh, UK, April 4–8, 2005, 2005. Springer.

[FA08] E. Feron and F. Alegre. Control software analysis, part I: Open-loop properties. Technical Report arXiv:0809.4812, arXiv.org, October 2008.

[FP80] G. Franklin and J. D. Powell. *Digital Control Of Dynamic Systems.* Addison-Wesley, 1980.

[Gou01] E. Goubault. Static analyses of the precision of floating point operations. In P. Cousot, editor, *Static Analysis: 8th International Symposium, SAS 2001,Paris, France, July 16-18, 2001 : Proceedings*, pages 234–257. Springer-Verlag, 2001.